HW5 ECE 719 Optimal systems

Spring 2016
Electrical engineering department
University of Wisconsin, Madison

Instructor: Professor B Ross Barmish

By

Nasser M. Abbasi

December 30, 2019

Contents

List of Figures

List of Tables

0.1 Problem 1

Barmish

ECE 719 – Homework Freudenstein

When one wishes to solve a set of nonlinear equations

$$f_i(x) = 0; \quad i = 1, 2, \ldots, N,$$

one can consider an optimization problem with cost function

$$J(x) = \sum_{i=1}^{N} f_i^2(x)$$

to be minimized.

(a) Explain the relationship between the optimization problem and the original nonlinear equation solving problem.

(b) For the two nonlinear functions of Freudenstein and Roth given by

$$f_1(x) = x_2 - x_1^3 + 5x_1^2 - 2x_1 - 13;$$

$$f_2(x) = x_2 + x_1^3 + x_1^2 - 14x_1 - 29,$$

generate some contours for $J(x)$ over the interesting region described by $|x_1| \leq 10;\ |x_2| \leq 50$.

(c) Write a program which implements the Polyak-Ribiere Algorithm (look up the iterative procedure) including your optimal line search method to minimize $J(x)$ using f_1 and f_2 above. In reporting your results, describe the performance of the algorithm from a variety of initial conditions x^0 including some illustrative iteration pathes superimposed on the J contours. Also indicate what line type of line search and stopping criterion you used.

Figure 1: problem 1 description

0.1.1 part(a)

Let $f_i(x) : \Re^n \to \Re$. We want to solve

$$f_i(\boldsymbol{x}) = 0 \qquad i = 1, 2, \cdots N \tag{1}$$

Which means finding $\boldsymbol{x}^*$ which makes value of $f_i(\boldsymbol{x}^*)$ be zero. If we consider the vector $F(\boldsymbol{x})$ of functions $f_i(\boldsymbol{x})$

$$F(\boldsymbol{x}) = \begin{pmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \\ \vdots \\ f_N(\boldsymbol{x}) \end{pmatrix}$$

Then the square of the Euclidean norm of $F(\boldsymbol{x})$ is

$$\|F(\boldsymbol{x})\|^2 = \sum_{i=1}^{N} f_i^2(\boldsymbol{x})$$

The minimum value of $\|F(\boldsymbol{x})\|$ is zero since it is a norm. Which is the same as $\|F(\boldsymbol{x})\|^2 = 0$. This means $F(\boldsymbol{x}) = 0$ occurs when $\|F(\boldsymbol{x})\|^2 = 0$. So the solution to (1) is the same $\boldsymbol{x}^*$ as finding the minimizer $\boldsymbol{x}^*$ which makes $\|F(\boldsymbol{x})\|^2$ minimum.

Therefore minimizing $J(x) = \|F(\boldsymbol{x})\|^2 = \sum_{i=1}^{N} f_i^2(\boldsymbol{x})$ will give the solution to (1). This is similar to finding least squares solution to set of linear equations, except now the set of equations $F(\boldsymbol{x})$ are non-linear in $\boldsymbol{x}$.

0.1.2 part b

$$\begin{aligned} f_1(x) &= x_2 - x_1^3 + 5x_1^2 - 2x_1 - 13 \\ f_2(x) &= x_2 + x_1^3 + x_1^2 - 14x_1 - 29 \end{aligned}$$

Hence

$$\begin{aligned} J(x) &= f_1^2(x) + f_2^2(x) \\ &= \left(x_2 - x_1^3 + 5x_1^2 - 2x_1 - 13\right)^2 + \left(x_2 + x_1^3 + x_1^2 - 14x_1 - 29\right)^2 \\ &= 2x_1^6 - 8x_1^5 + 2x_1^4 - 80x_1^3 + 12x_1^2x_2 + 12x_1^2 - 32x_1x_2 + 864x_1 + 2x_2^2 - 84x_2 + 1010 \end{aligned} \tag{1}$$

$J(x)$ is non-linear function. The above is the $\|F(\boldsymbol{x})\|^2$ where now $F(\boldsymbol{x}) = \begin{pmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \end{pmatrix}$. We will use optimization to find the solution $\boldsymbol{x}$ to $F(\boldsymbol{x}) = 0$ by finding the minimizer of (1). The solution will turn out to be

$$\boldsymbol{x}^* = (x_1 = 4, x_2 = 5)$$

At this point $J(x^*) = 0$ and also $f_1(x^*) = 0$ and $f_2(x^*) = 0$. So the above is the true solution to $f_i(x) = 0$. But there is also another local minimum close to it located at

$$\boldsymbol{x} = (x_1 = -0.8968, x_2 = 11.4128)$$

where here $J(x) = 48.98$ and not zero. At this second local minimum, the corresponding values for f_i are $f_1(\boldsymbol{x}) = 4.949$ and $f_2(\boldsymbol{x}) = -4.949$. These were found by running the conjugate gradient algorithm with Polyak-Ribiere stepping as given below.

The following is contour plot of the full range given in the problem statement, showing there are two local minimums, one around point $x_1 = 4.5, x_2 = 8$ and another around $x_1 = -1.3, x_2 = 10$

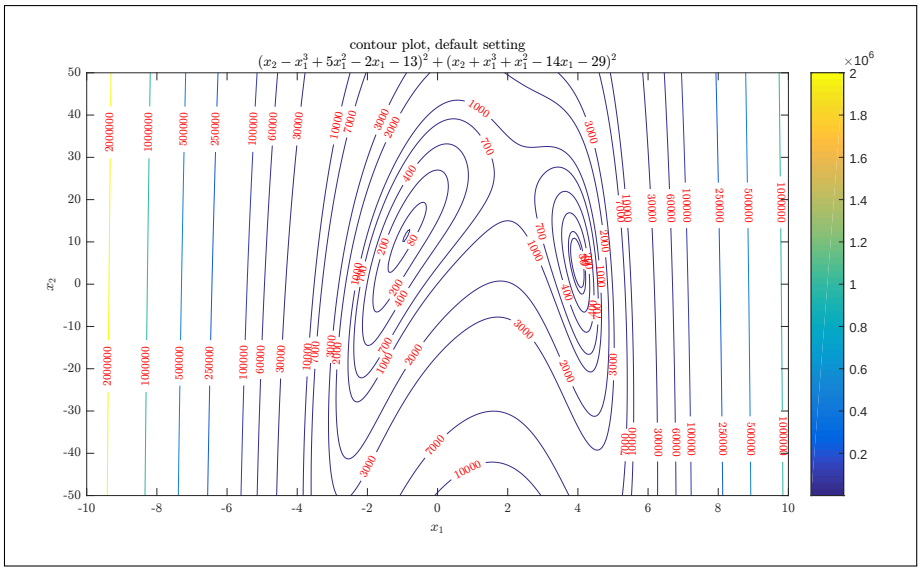


Figure 2: contour plot, full range

This is zoomed version of the above to show more clearly the area around the variations

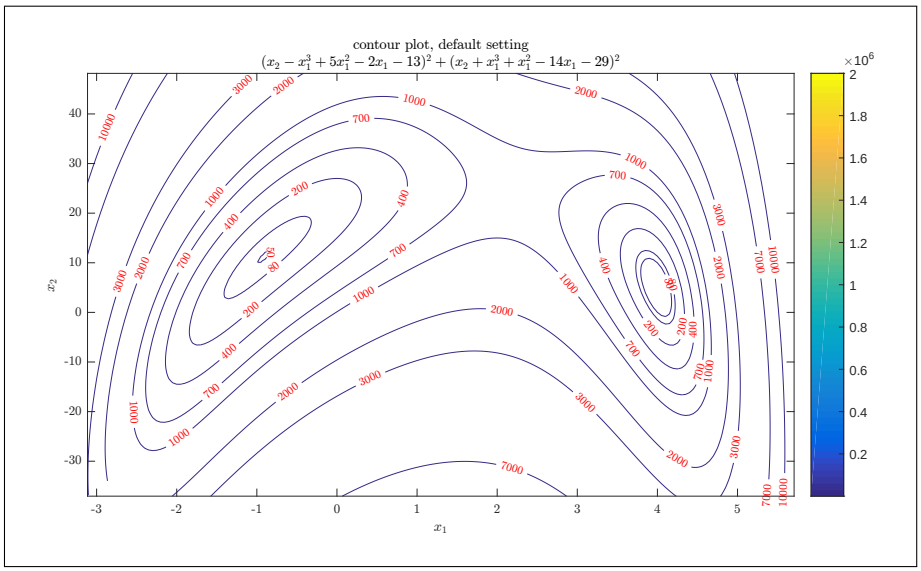


Figure 3: contour plot, zoomed version

This is filled contour version of the above.

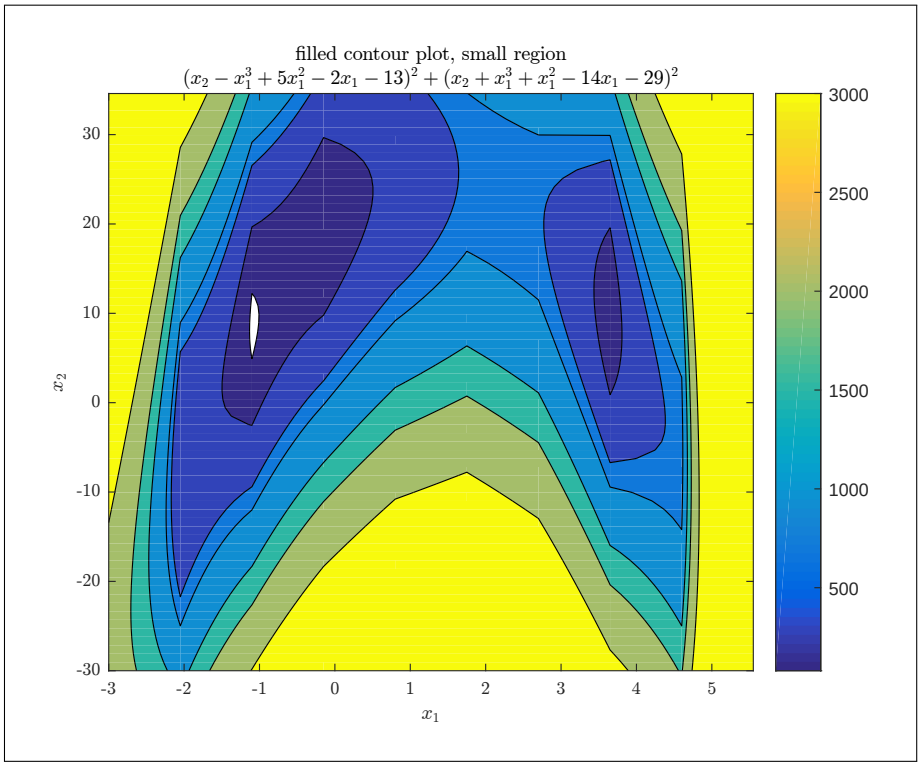


Figure 4: contour plot, filled zoomed version

This is 3D plot of the function $J(x)$

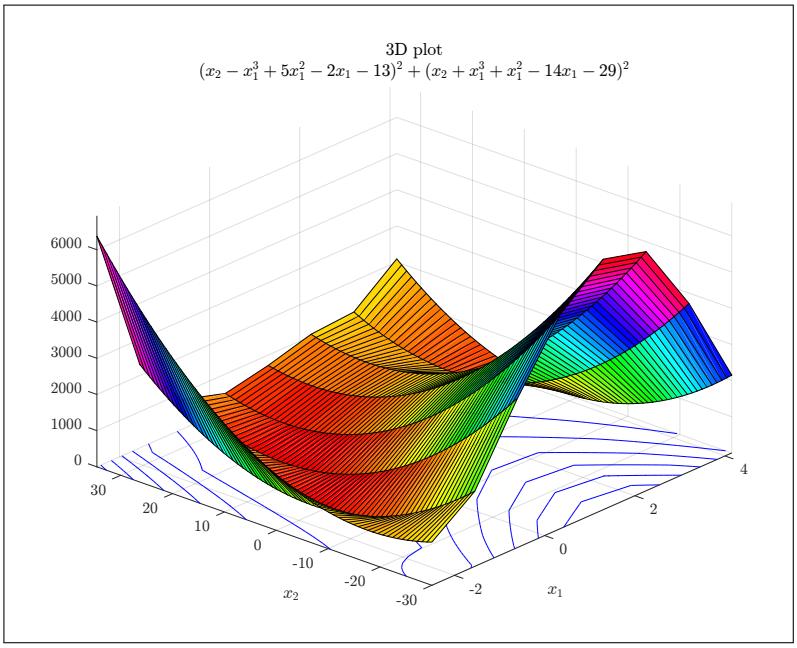


Figure 5: contour plot, filled zoomed version

0.1.3 part(c)

A Matlab program is given in the appendix which implements Polyalk-Ribiere (and it also supports Fletcher-Reeves). The result is given below, with discussion following each result. One result also shows an interesting difference found between Polyalk-Ribiere and Fletcher-Reeves when starting from some random found u^0 point. In all runs, Matlab fminsearch was also used to compare the minimizer found. In some cases, this algorithm found the same minimum as Matlab's fminsearch, and in other cases it did not.

The result shows each point u^k visited with the value of β_k and α_k found, and the value of the objective function and the gradient at each step.

For the line search, golden section search was used to find α_k with maximum step size $H_{\max} = 1$. The stopping criteria used for all these runs is $|\nabla J(u)| \leq 0.001$.

The algorithm

The following is the outline of general algorithm expressed as pseudo code.

Algorithm 1 Conjugate gradient using Polyalk-Ribiere or Fletcher-Reeves

procedure CONJUGATE_GRADIENT
  ▷ Initialization
  $\epsilon \leftarrow$ minimum convergence limit on $\|\nabla J(u)\|$
  $k \leftarrow 0$
  $u \leftarrow u^0$
  $max\_iterations \leftarrow$ max iterations allowed
  $g_{current} \leftarrow \nabla J(u)$
  $v_{current} \leftarrow -g_{current}$
  while $\|g_{current}\| > \epsilon$ **do**
    ▷ do line search, using golden section, maximum step size is one
    $\alpha \leftarrow \min_\alpha \tilde{J}(\alpha) = J(u + \alpha\, v_{current})$
    $g_{previous} \leftarrow g_{current}$
    $u \leftarrow u + \alpha v_{current}$
    $g_{current} \leftarrow \nabla J(u)$
    if Fletcher-Reeves **then**
      $\beta \leftarrow \frac{\|g_{current}\|^2}{\|g_{previous}\|^2}$
    else if Polyalk-Ribiere **then**
      $\beta \leftarrow \frac{g_{current}^T (g_{current} - g_{previous})}{\|g_{previous}\|^2}$
    end if
    $v_{current} \leftarrow -g_{current} + \beta\, v_{current}$
  end while
end procedure

Figure 6: Conjugate gradient using Polyalk-Ribiere or Fletcher-Reeves

Test 1 starting from $(-5.49, 23.05)$

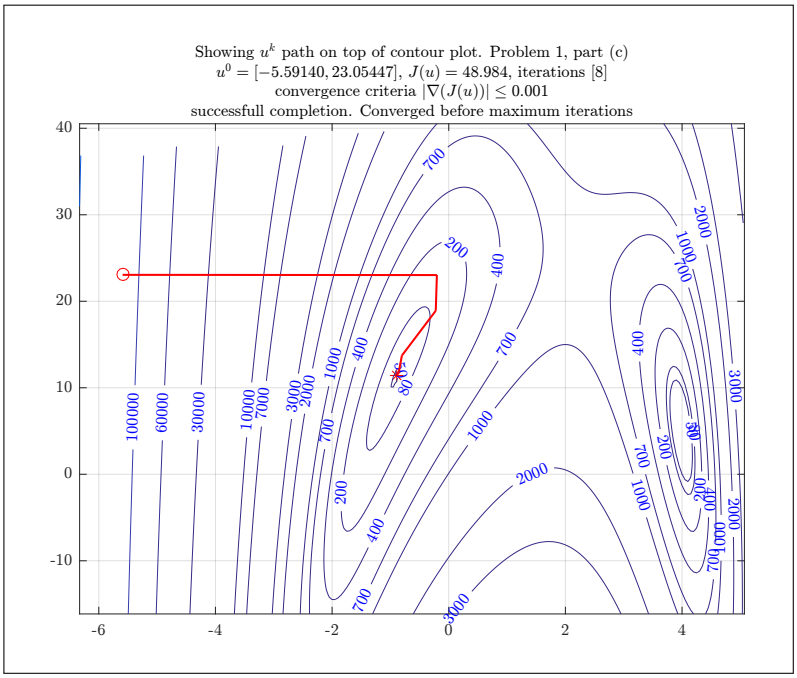


Figure 7: test case 1, problem 1, part c

Matlab fminsearch found $J(-0.8968, 11.4128) = 48.9843$. This program found $J(-0.8968, 11.4128) = 48.9843$. since $u^* = (4, 5)$ we see that the search did not find u^* but found the other local minimum near it, since the search was started from a point closer to the second one. Also we see Matlab fminsearch result matched our result. So this is good. It took 8 steps. We also notice that $\nabla J\left(x^k\right)$ increased at one step (step 3) during the search. This is indication that this is not a quadratic function (which we already know this), but $\nabla J\left(x^k\right)$ started to decrease again after that.

k	x^k	$J\left(x^k\right)$	$\left\|\nabla J\left(x^k\right)\right\|$	α_k	β_k
1	$(-5.59, 23.05)$	129231.19	116683.427	0.000046	-0.000003
2	$(-0.20, 23.028)$	122.99	15.12	0.273	86.62
3	$(-0.2227, 18.9)$	91.83	140.56	0.003958	0.3535
4	$(-0.8, 13.72)$	52.15	39.059	0.00394	0.4082
5	$(-0.855, 11.885)$	49.16	12.1855	0.00236	0.05
6	$(-0.896, 11.436)$	48.98	0.583	0.00238	0.0094
7	$(-0.8968, 11.4129)$	48.98	0.00545	0.002095	0.00123
8	$(-0.8968, 11.4128)$	48.98	0.000007	0.000000	0.000000

Test 2 starting from $(5.8, 35.89)$

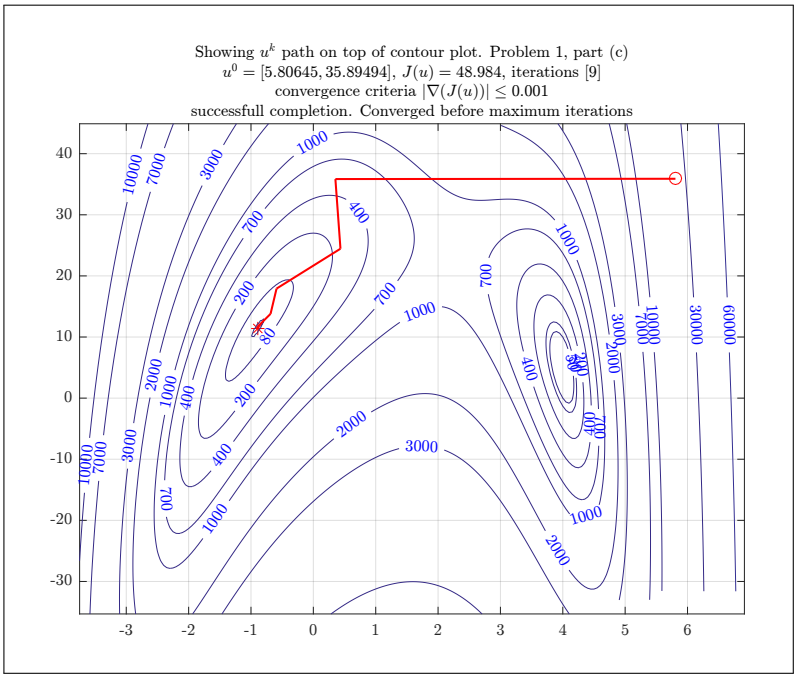


Figure 8: test case 2, problem 1, part c

Matlab fminsearch found $J(-0.8968, 11.4128) = 48.9843$. This program found $J(-0.8968, 11.4128) = 48.9843$. since $u^* = (4, 5)$ we see again that the search did not find u^* but found the other local minimum. Also we see Matlab fminsearch result matched our result. So this is good. It took 9 steps. We also notice that $\nabla J\left(x^k\right)$ increased at one step (step 3) during the search.

k	x^k	$J\left(x^k\right)$	$\left\|\nabla J\left(x^k\right)\right\|$	α_k	β_k
1	(5.806, 35.895)	24303.88	32066.72	0.000170	0.000011
2	(0.353, 35.847)	520.53	49.582	0.2299	36.96
3	(0.435, 24.448)	238.41	301.265	0.00356	0.28
4	(−0.59, 17.92)	71.95	69.317	0.0079	1.3363
5	(−0.686, 13.789)	53.18	52.828	0.0028	0.1788
6	(−0.883, 11.7991)	49.08	8.1969	0.00295	0.06401
7	(−0.895, 11.4284)	48.98	0.4961	0.00193	0.00629
8	(−0.896801, 11.412913)	48.98	0.003106	0.002634	0.000101
9	(−0.896805, 11.412779)	48.98	0.000000	0.000000	0.000000

Test 3 starting from (5.59, –19.55)

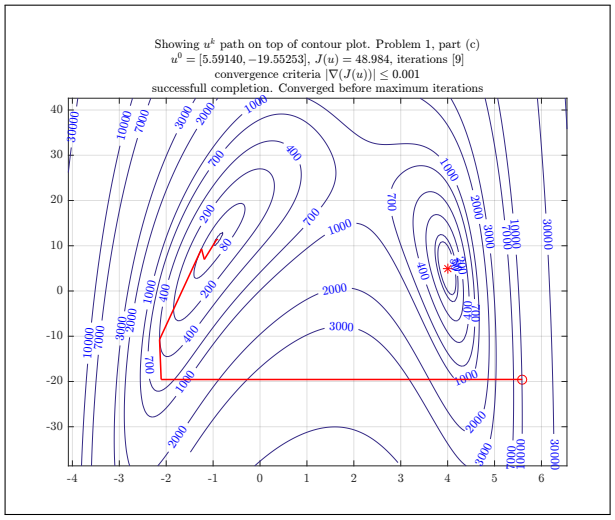


Figure 9: test case 3, problem 1, part c

Matlab fminsearch found the true minimum $J(4,5) = 0$. This program did not do as well, and went for the second local minimum at $J(-0.8968, 11.4128) = 48.9843$, which has the corresponding solution $f_1(x) = 4.9490, f_2(x) = -4.9490$.

One surprising thing to note, is that Matlab fminsearch uses simplex method according to the help. But this problem is not linear. It turns out that Matlab fminsearch uses a modified version of simplex method, called the Nelder-Mead simplex (direct search). It seems to do better than algorithm implemented in this problem. But in the next test case, we will see that this algorithm evens the score with Matlab's and in the next test case it is we who will do better.

It took 9 steps. Again as before, $\nabla J\left(x^k\right)$ increased at one step during the search (at step 3 also).

k	x^k	$J\left(x^k\right)$	$\left\|\nabla J\left(x^k\right)\right\|$	α_k	β_k
1	(5.5914, –19.553)	10150.82	19380.2	0.000397	–0.000023
2	(–2.107, –19.566)	585.39	41.5373	0.2127	423.717
3	(–2.142, –10.731)	401.83	854.79	0.001138	–0.213
4	(–1.247, 9.303)	79.34	264.025	0.000619	–0.14389
5	(–1.1875, 6.974)	57.85	46.1832	0.00822	–0.2476
6	(–0.9218, 11.436)	49.30	24.1288	0.001065	–0.02258
7	(–0.9046, 11.29)	48.99	0.56707	0.0389	–0.0926
8	(–0.8969, 11.4131)	48.98	0.05981	0.001129	–0.000415
9	(–0.896806, 11.412772)	48.98	0.000025	0.000000	0.000000

Test 4 starting from (7.43472, 16.05058)

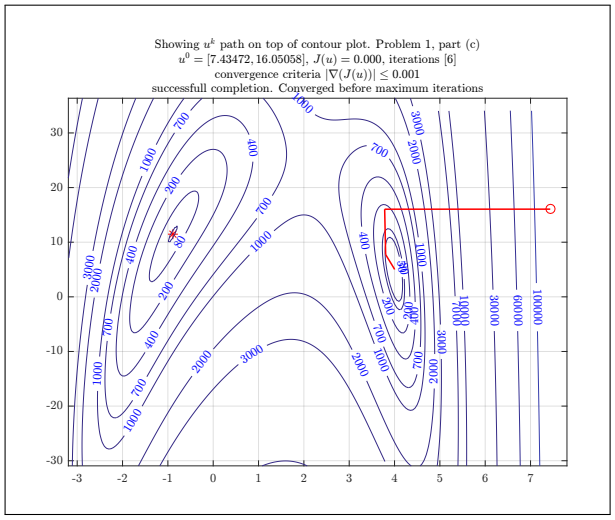


Figure 10: test case 4, problem 1, part c

Matlab fminsearch here did not find the true minimum $J(4,5) = 0$ while this algorithm did.

It took 6 steps only. Again as before, $\nabla J\left(x^k\right)$ increased at one step during the search (at step 2).

k	x^k	$J\left(x^k\right)$	$\left\|\nabla J\left(x^k\right)\right\|$	α_k	β_k
1	(7.435, 16.05)	143368.39	143787.679	0.000025	0.000002
2	(3.78, 16.04)	172.57	30.799	0.2696	200.218
3	(3.84, 7.74)	44.74	435.9738	0.000433	0.00797
4	(3.999778, 5.066770)	0.01	3.455556	0.001349	0.009287
5	(3.999989, 5.000154)	0.00	0.031875	0.000336	−0.000038
6	(4.000000, 5.000000)	0.00	0.000001	0.000000	0.000000

Test 5 starting from (3.809, −8.46)

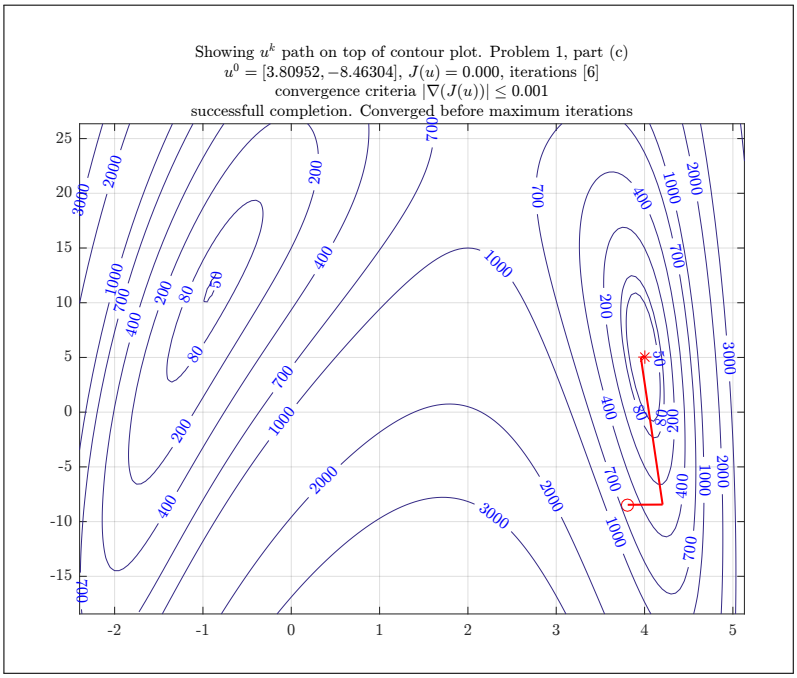


Figure 11: test case 5, problem 1, part c

Here both Matlab fminsearch and this algorithm, found the true minimum.

It took 6 steps only. Again as before, $\nabla J\left(x^k\right)$ increased at one step during the search (at step 3).

k	x^k	$J\left(x^k\right)$	$\left\|\nabla J\left(x^k\right)\right\|$	α_k	β_k
1	(3.809524, –8.463035)	580.29	1386.28	0.000285	0.000784
2	(4.204487, –8.444322)	267.86	40.2297	0.33335	14.037757
3	(3.958554, 4.969723)	3.20	150.186235	0.000278	–0.009224
4	(3.997429, 5.127561)	0.02	1.447732	0.022782	0.258685
5	(4.000078, 5.000400)	0.00	0.316225	0.000273	0.000175
6	(4.000000, 5.000004)	0.00	0.000057	0.000000	0.000000

Test 6 (first strange one) starting from (6.63594, –14.29961)

This test case and the second one are pathological cases, in the sense that this algorithm did find the true minimum, but the path taken headed first to the second local minimum and was very close to it, before turning and going to the true minimum at (4, –5). At this time, I am not able to explain this and more time needed to investigate. It does however find the true minimum eventually, so this is good result even if the path taken looks very strange compared to all the other tests above. The main difference between this test case and the last ones, is that here the objective function $J(x)$ increased at one point during the search (at step 6 as shown below).

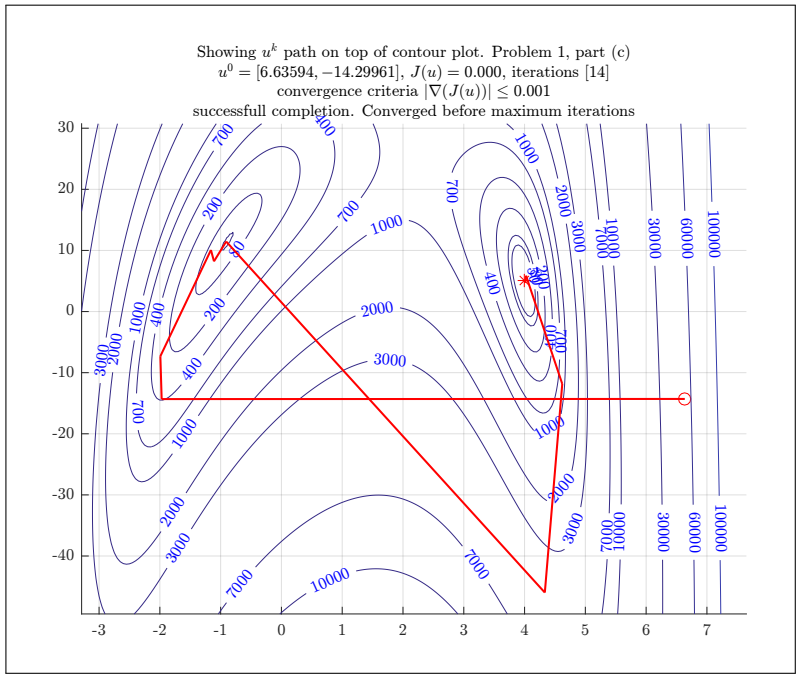


Figure 12: test case 6, Polyak-Ribiere, problem 1, part c

Here both Matlab fminsearch and this algorithm, found the true minimum.

It took 14 steps. Here $\nabla J\left(x^k\right)$ increased and decreased more than one time during the search.

k	x^k	$J\left(x^k\right)$	$\left\|\nabla J\left(x^k\right)\right\|$	α_k	β_k
1	(6.63594, –14.29961)	52701.77	67823.57	0.000127	0.000002
2	(–1.970382, –14.321801)	393.95	31.646	0.221630	385.651
3	(–1.991739, –7.308131)	282.95	621.5254	0.001424	–0.217509
4	(–1.159618, 10.073263)	67.36	199.4501	0.000696	–0.132737
5	(–1.109423, 8.218728)	53.56	31.5552	0.009140	–0.241872
6	(–0.908598, 11.459289)	49.08	13.239	0.662964	22576.86
7	(4.328897, –45.933286)	4299.56	1995.887	0.000000	–0.009991
8	(4.333267, –45.980644)	4299.50	1975.7426	0.001732	3.309271
9	(4.620182, –11.840013)	883.28	2743.2211	0.000271	–0.051462
10	(4.024522, 5.862406)	3.99	149.444	0.000338	–0.154415
11	(4.012227, 4.726667)	0.22	28.564	0.000532	–0.010341
12	(4.000032, 5.002778)	0.00	0.299	0.000518	–0.004453
13	(4.000001, 4.999987)	0.00	0.00134	0.000550	0.001374
14	(4.000000, 5.000000)	0.00	0.000002	0.000000	0.000000

The above was re-run again, starting from the same u^0, but now using Fletcher-Reeves formula. The result was surprising. Now the algorithm did not show the strange path as above, however, it also did not find the true minimum at (4, –5) and instead went for the

second local minimum as shown below. This shows, at least in this test, that Polyak-Ribiere formula did a better job, even though it took more steps.

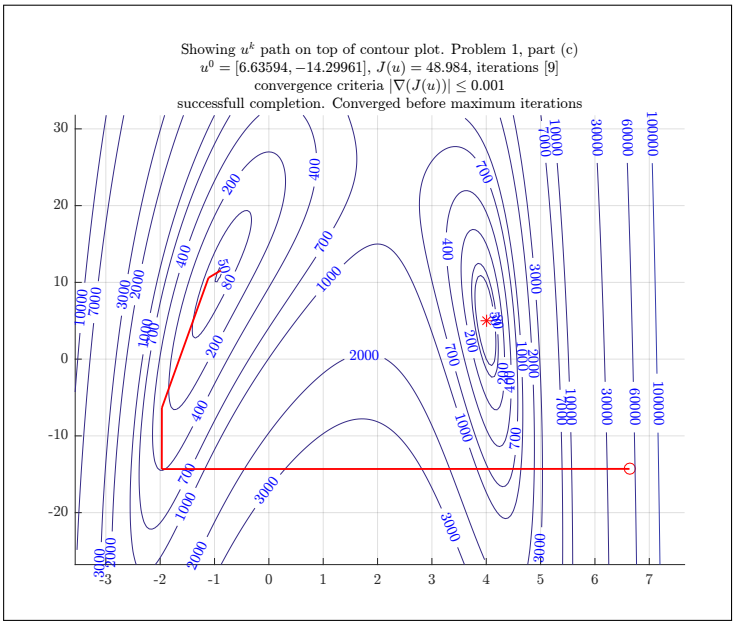


Figure 13: test case 6, using Fletcher-Reeves, problem 1, part c

k	x^k	$J\left(x^k\right)$	$\left\|\nabla J\left(x^k\right)\right\|$	α_k	β_k
1	(6.63594, −14.29961)	52701.77	67823.57	0.000127	0.000000
2	(−1.970382, −14.321801)	393.95	31.646	0.2519	393.1282
3	(−1.968895, −6.351520)	267.83	627.462	0.001362	0.07595
4	(−1.111164, 10.592228)	63.10	172.916	0.001001	0.000010
5	(−0.890514, 11.528839)	48.99	0.5567	0.001137	0.03014
6	(−0.889896, 11.528704)	48.99	0.0967	0.888831	202.77
7	(−0.893567, 11.441590)	48.99	1.3763	0.001469	0.000012
8	(−0.896818, 11.412476)	48.98	0.00481	0.001101	0.002681
9	(−0.896823, 11.412476)	48.98	0.000249	0.000000	0.000000

Test 7 (second strange one) starting from (0.5837, −46.595)

This test case also showed difference between Polyak-Ribiere and Fletcher-Reeves.

With Polyak-Ribiere, it found the same minimum as Matlab fminsearch using a strange path where $J(u)$ did increase at one point before decreasing again.

However, it did a better job than Fletcher-Reeves.

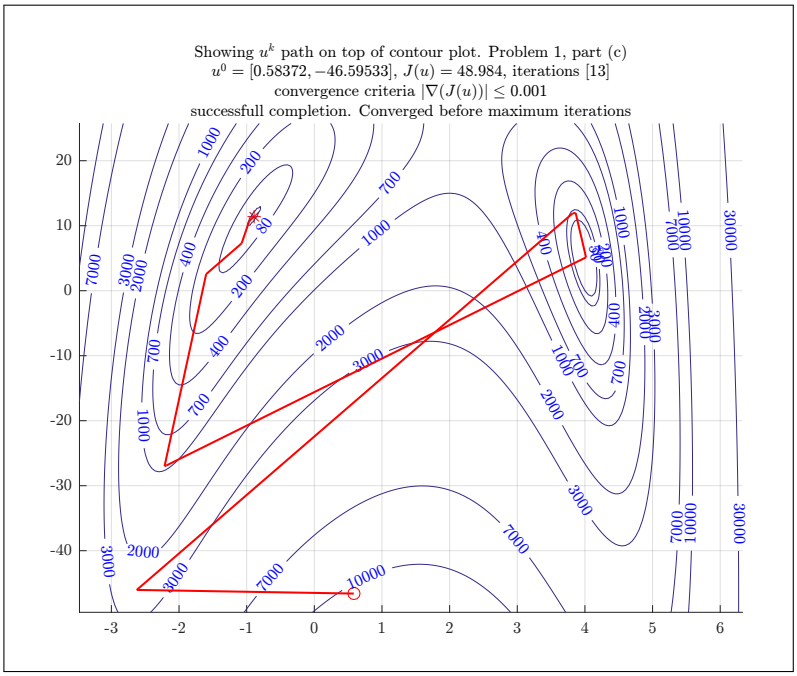


Figure 14: test case 7, Polyak-Ribiere, problem 1, part c

Here both Matlab fminsearch and this algorithm did not find the true minimum.

It took 13 steps. Here $\nabla J\left(x^k\right)$ increased and decreased more than one time during the search. Also $J(u)$ increased during the search before decreasing again.

k	x^k	$J\left(x^k\right)$	$\left\|\nabla J\left(x^k\right)\right\|$	α_k	β_k
1	(0.583720, −46.595330)	10438.38	1656.968	0.001966	0.003862
2	(−2.624791, −46.035172)	2450.06	103.007	0.564906	2.002
3	(3.819328, 11.909200)	72.20	148.967	0.000257	−0.1194
4	(3.863288, 11.957784)	69.31	28.774	0.163101	7.727
5	(4.016286, 5.132002)	0.67	70.2157	0.098572	18.327
6	(−2.188772, −26.898544)	959.12	336.852	0.000017	−0.183
7	(−2.213801, −26.997880)	958.16	255.113	0.025963	3.785
8	(−1.599015, 2.551447)	123.88	387.315	0.001099	0.179
9	(−1.074840, 7.277357)	57.59	60.1637	0.004433	0.517
10	(−0.961876, 10.715217)	49.45	22.635	0.001854	−0.0502
11	(−0.895537, 11.456508)	48.99	1.2014	0.002193	−0.01161
12	(−0.896851, 11.412270)	48.98	0.014138	0.002174	0.000948
13	(−0.896805, 11.412779)	48.98	0.000013	0.000000	0.000000

0.1.4 Appendix. Source code for problem 1

```
1 function nma_HW5_problem_1_part_b
2 %Solves problem 1, part b, HW5
```

```
%ECE 719, UW Madison, Spring 2016
%
%

close all; clc;
cd(fileparts(mfilename('fullpath')));

%reset(0);
xlimits   = [-10 10];  %x limits, for plotting, change as needed
ylimits   = [-50 50]; %y limits, for plotting, change as needed
myTitle   = ...
 '$$(x_2-x_1^3+5x_1^2-2x_1-13)^2+(x_2+x_1^3+x_1^2-14x_1-29)^2$$';
[u1,u2,z] = nma_makeContourData(0.05,xlimits,ylimits);

figure(1);
v =[50 80 200 400 700 10^3 2*10^3 3*10^3 7*10^3 10^4 ...
           3*10^4 6*10^4 10^5 2.5*10^5 ...
           5*10^5 10^6 2*10^6];
[C,h]    = contour(u1,u2,z,v);
colorbar
%[C,h]    = contour(u1,u2,z);

clabel(C,h,v,'Fontsize',7,'interpreter','Latex','Color','red');
nma_setMyLabels('$$x_1$$','$$x_2$$',...
    {'\makebox[4in][c]{contour plot, default setting}',...
    sprintf('\\makebox[4in][c]{%s}',myTitle)});

figure();
xlimits   = [-3 6];  %x limits, for plotting, change as needed
ylimits   = [-30 35]; %y limits, for plotting, change as needed
[X,Y,Z] = nma_makeContourData(.95,xlimits,ylimits);
v        =[80 300 700 900 1500 2000 3000];
[C,h]    = contourf(X,Y,Z,v);
colorbar;
nma_setMyLabels('$$x_1$$','$$x_2$$',...
    {'\makebox[4in][c]{filled contour plot, small region}',...
    sprintf('\\makebox[4in][c]{%s}',myTitle)});

figure();
xlimits   = [-2.5 5];  %x limits, for plotting, change as needed
ylimits   = [-30 30]; %y limits, for plotting, change as needed
[X,Y,Z] = nma_makeContourData(.95,xlimits,ylimits);
%surfl(X,Y,Z);
surf(X,Y,Z);
colormap(hsv);
%view([154,46]);
hold on;
```

```
contour(X,Y,Z,'Linecolor',[0 0 1]);
nma_setMyLabels('$$x_1$$','$$x_2$$',...
    {'\makebox[4in][c]{surf plot}',...
    sprintf('\\makebox[4in][c]{%s}',myTitle)});

end
```

```
function nma_HW5_problem_1_part_c()
%finds the min value of
%
%    J(x) = f1^2 + f2^2 where
%    f1 = x2-x1^3+5*x1^2-2x1-13
%    f2=x2+x1^3+x1^2-14x1-29
%
% over range x1=-10..10 and x2=-50..50 using steepest descent
%
% ECE 719, Spring 2016
% Matlab 2015a
%Nasser M. Abbasi

if(~isdeployed)
    baseFolder = fileparts(which(mfilename));
    cd(baseFolder);
end

close all;
set(groot,'defaulttextinterpreter','Latex');
set(groot, 'defaultAxesTickLabelInterpreter','Latex');
set(groot, 'defaultLegendInterpreter','Latex');

%paramters, change as needed
%select the algorithm to use. Either 'conjugate gradient'
%or 'steepest descent'
METHOD       = 'conjugate gradient';

DO_GUI       = false;    %set to true to get input from GUI
DO_ANIMATE   = true;     %set to true to see animation
DO_GIF       = false;    %set to true to make animation gif
DO_3D        = false;    %if we want to show 3D search path.
xlimits      = [-20 20];   %x limits, for plotting
ylimits      = [-90 90];   %y limits, for plotting
del          = 0.05;     %grid size, used for making meshgrid
fixed_levels = [50 80 200 400 700 10^3 2*10^3 3*10^3 ...
                7*10^3 10^4 3*10^4 ...
                6*10^4 10^5 2.5*10^5 5*10^5 10^6 2*10^6];
CONTOUR_LINES_AUTO = 'fix'; %set to 'auto', to see matlab contour
%                     %set to 'full' to see each step level set
%                     %set to 'limited' to see every other level
```

```
%                       %set to 'fix' to use pre-specificed
%

%-------------------------------------------------------
%These are the options struct used by call to
%optimization function
%opt.u  = [6.63594;-14.29961];   %starting guess x-coordinate
%selection of tough ones: This gives very different
%                                result from fletcher and polak.
%opt.u  = [6.63594;-14.29961];   %starting guess x-coordinate
opt.u = [0.58372;-46.59533];   %starting guess x-coordinate

opt.MAX_ITER        = 10^4; %maximum iterations allowed
opt.STEP_SIZE       = -1;  %step size. set to -1 to use optimal
opt.objectiveFunc   = @objectiveFunc; %see function definition
opt.gradientFunc    = @gradientFunc;  %see function definition
opt.gradientNormTol = 0.001;  %used to determine when converged
opt.hessian         = @hessian_func;  %see function definition
opt.accumulate      = true;
opt.stop_on_oscillation = false;

%----------------------------------------
%data
[u1,u2,z] = nma_makeContourData(del,xlimits,ylimits);
figure();
if DO_GUI  %check if GUI input is asked for, if so, wait for user
    plot(0,0);
    xlim(xlimits); ylim(ylimits);
    hold on;
    [x,y] = ginput(1);
    opt.u=[x;y];
end

%Find the minumum using Matlab build-in, in order to
%compare with in plot
optimalValue = fminsearch(opt.objectiveFunc, opt.u);
objectiveAtOptimal = objectiveFunc(optimalValue);
fprintf('Matlab found J(%5.4f,%5.4f)=%5.4f\n',optimalValue(1),...
                            optimalValue(2),objectiveAtOptimal);

%mark location of minimum found by fminsearch on plot
hold on;
plot(optimalValue(1),optimalValue(2),'*r');

%plot starting point
plot(opt.u(1),opt.u(2),'or');
xlim(xlimits); ylim(ylimits);
```

```
grid;
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

%make the call to find search path.
if strcmp(METHOD,'steepest descent')
   [status , pts,levelSets, gradientNormTol,steps] = ...
                                      nma_steepest_descent(opt);
else
   [status,pts,levelSets, gradientNormTol,steps,betaK]=...
                                         nma_polyak_ribiere(opt);
end
fprintf('HW5 found J(%5.4f,%5.4f)=%5.4f\n',pts(end,1),...
                                      pts(end,2),levelSets(end));

%check if search was success or not.
switch status
    case 0, status = ...
'successfull completion. Converged before maximum iterations';
    case 1, status = ...
'failed to converge before maximum iterations due to oscillation';
    case 2, status = ...
'failed to converge before maximum iterations';
end

%use output from above call to make the plots
switch CONTOUR_LINES_AUTO
    case 'auto',
     [C,h] =contour(u1,u2,z,'Linecolor',[0 0 1],'LineWidth',0.1);
    case 'limited',
      lev   = round(length(levelSets)/20);
      %[C,h] = contour(u1,u2,z,levelSets(1:lev:end),'Fill','off');
      %[C,h]     = contourf(u1,u2,z,levelSets(1:lev:end));
      [C,h]     = contour(u1,u2,z,levelSets(1:lev:end));
        %colormap(hsv);
        %colorbar;
        %'Linecolor',[0 0 1],'LineWidth',.2);
    case 'full'
      [C,h] = contour(u1,u2,z,levelSets,'LineWidth',.2);
      clabel(C,h,'Fontsize',8,'interpreter','Latex',...
                                                 'Color','blue');
    case 'fix'
        [C,h] = contour(u1,u2,z,fixed_levels);
        h.LineWidth = .1;
        %h.LineColor  = [190/255 190/255 190/255];
        clabel(C,h,fixed_levels,'Fontsize',8,...
                         'interpreter','Latex','Color','blue');
end
```

```
136
137 %animate the steepest descent search
138 if length(pts(:,1))>1
139     filename = 'anim.gif';
140     for k=1:length(pts)-1
141         %draw line between each step
142         %skip case if 'full' mode or if too many points.
143         %if (opt.STEP_SIZE == -1 || ...
144                    %strcmp(CONTOUR_LINES_AUTO,'limited') || ...
145         %strcmp(CONTOUR_LINES_AUTO,'auto')||length(pts)<100 )
146         line([pts(k,1),pts(k+1,1)],[pts(k,2),pts(k+1,2)],...
147                                     'LineWidth',1,'Color','red');
148         %end
149         %plot([pts(k,1),pts(k+1,1)],[pts(k,2),pts(k+1,2)],'.r');
150         if DO_ANIMATE
151             drawnow;
152             if DO_GIF
153                frame = getframe(1);
154                im = frame2im(frame);
155                [imind,cm] = rgb2ind(im,256);
156                if k ==1
157                   imwrite(imind,cm,filename,'gif','Loopcount',0);
158                else
159                   if mod(k,4)==0
160                      imwrite(imind,cm,filename,'gif',...
161                                             'WriteMode','append');
162                   end
163                end
164             end
165         end
166         title(format_plot_title(...
167 'Showing $u^k$ path on top of contour plot. Problem 1, part (c)',...
168            opt,pts,k,status),'FontSize', 8);
169     end
170 end
171 title(format_plot_title(...
172 'Showing $u^k$ path on top of contour plot. Problem 1, part (c)',...
173  opt,pts,size(pts,1),status),'FontSize', 8);
174
175
176 %plot J(x) changes
177 figure();
178 stairs(levelSets);
179 %stem(levelSets,'ro');
180 grid;
181 set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
182 title(format_plot_title(...
```

```
            'Showing $J(u^k)$ progress. Problem 1, part (b)',...
    opt,pts,size(pts,1),status),'FontSize', 8);
xlabel('step number');
ylabel('value of objective function');

%Plot gradient change
figure();
stairs(gradientNormTol);
%stem(levelSets,'ro');
grid;
title(format_plot_title(...
   'Showing $|\nabla J(u^k)|$ progress. Problem 1, part (c)',...
    opt,pts,size(pts,1),status),'FontSize', 8);

xlabel('step number'); ylabel('Norm of gradient');
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

%Plot betaK
if strcmp(METHOD,'conjugate gradient')
figure();
stem(betaK);
grid;
title(format_plot_title(...
        'Showing $\beta(k)$ progress. Problem 1, part (c)',...
    opt,pts,size(pts,1),status),'FontSize', 8);
xlabel('step number'); ylabel('$\beta_k$');
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
end

%Plot alpha
if strcmp(METHOD,'conjugate gradient')
figure();
stem(steps);
grid;
title(format_plot_title(...
        'Showing $\alpha(k)$ progress. Problem 1, part (c)',...
    opt,pts,size(pts,1),status),'FontSize', 8);
xlabel('step number'); ylabel('$\alpha_k$');
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
end

format long g;
fprintf('x1\t\t\tx2\t\t\tJ(x1,x2)\tgrad(J(u))\talpha\t\tbeta\n');
for i=1:length(steps)
    fprintf('%7.6f\t%7.6f\t%5.2f\t%7.6f\t%7.6f\t%7.6f\n',...
              pts(i,1),pts(i,2),...
              levelSets(i),gradientNormTol(i),steps(i),betaK(i));
```

```
end

f1 =@(X1,X2) X2-X1^3+5*X1^2-2*X1-13;
f2 = @(X1,X2) X2+X1^3+X1.^2-14*X1-29;

fprintf('f1(x)=%5.4f,f1(x)=%5.4f\n',...
         f1(pts(end,1),pts(end,2)),f2(pts(end,1),pts(end,2)));

end
%-------------------------
%Evaluate J(u) at u
function f = objectiveFunc(u)
X1  = u(1);
X2  = u(2);
f1 = X2-X1.^3+5*X1.^2-2*X1-13;
f2 = X2+X1.^3+X1.^2-14*X1-29;
f = f1.^2+f2.^2;
end
%---------------------
%Evaluate grad(J(u)) at u
function g = gradientFunc(u)
x1  = u(1);
x2  = u(2);
g1=2*(3*x1^2 + 2*x1 - 14)*(x1^3 + x1^2 - 14*x1 + x2 - 29) +...
 2*(3*x1^2 - 10*x1 + 2)*(x1^3 - 5*x1^2 + 2*x1 - x2 + 13);
g2=12*x1^2 - 32*x1 + 4*x2 - 84;
g=[g1;g2];
end
%--------------------------
%set title
function formatted_title = format_plot_title(main_title,opt,pts,k,status)
formatted_title = {sprintf('\\makebox[5in][c]{%s}',main_title),...
  sprintf('\\makebox[5in][c]{$u^0=[%6.5f,%6.5f]$,  $J(u)=%3.3f$, iterations [$%d$]}',...
  opt.u(1),opt.u(2),norm(opt.objectiveFunc(pts(k,:))),k),...
  sprintf('\\makebox[5in][c]{convergence criteria $| \\nabla(J(u)) | \\leq  %1.3f $}',...
  opt.gradientNormTol),...
  sprintf('\\makebox[5in][c]{%s}',status)};
end
%---------------------
%Evaluate Hessian(J(u)) at u (not used, for practice)
function g = hessian_func(u)
x1  = u(1);
x2 = u(2);

g11=2*(6*x1 - 10)*(x1^3 - 5*x1^2 + 2*x1 - x2 + 13) + ...
    2*(3*x1^2 - 10*x1 + 2)^2 ...
    + 2*(3*x1^2 + 2*x1 - 14)^2 + ...
```

```
    2*(6*x1 + 2)*(x1^3 + x1^2 - 14*x1 + x2 - 29);
g12=24*x1 - 32;
g21=24*x1 - 32;
g22=4;
g=[g11,g12;g21,g22];
end
```

```
function [status,pointsFound,levelSets,gradientNormTol,steps,betaK]= ...
      nma_polyak_ribiere(opt)
% This function performs conjugate gradient search
% starting from a point looking for point which minimizes a
% function. Supports multi-variable function. It needs handle
% of the funtion and handle to the gradient. It reurns all
% points visited in the search. This supports Fletcher-Reeves
% and Polyalk-Ribiere
%
% Typical use of this function is as follows:
%
%  opt.field = ...%fill in each field of the struct.
%  [pointsFound,levelSets,gradientNormTol,steps]  =
%                                      nma_steepest_descent(opt);
%  [C,h]    = contour(.....,levelSets);
%
% INPUT fields in opt struct are:
% ======
% u                 vector of coordinates starting guess
% MAX_ITER          an integer, which is the maximium iteration
%                   allowed before giving up the search.
% gradientNormTol   small floating point number. The tolerance
%                   to use to decide when to stop the search.
%                   Example 0.001
% stepSize          A floating point number, which is the step
%                   size to take. If stepSize=-1 then an optimal
%                   step size is found and used
%                   at each step using golden section line search.
% objectiveFunc     handle to the objective function, which
%                   accepts a row vector, that contain [x y]
%                   coordinate of the point and return the
%                   numerical value of objectiveFunc at this point.
% gradientFunc      handle to the gradiant of f. Same input
%                   and output as objectiveFunc
% accumulate        flag. If true, then all points u^k and J(u)
%                   at each are collected during search. Else they
%                   are not.
% stop_on_oscillation  flag. Set to true to stop when objective
%                  function detected to be increasing. Else set
%                  to false if you do not want to stop when J(u)
%                  increases at any point
```

```
42 %
43 % OUTPUT:
44 % =======
45 % status           can be 0,1 or 2.
46 %                  0 means success, It converged before MAX_ITER
47 %                  was reached.
48 %                  1 means failed, did not converge due to
49 %                  oscillation, which can happen when step size
50 %                  is too large. When oscillation detected, the
51 %                  search will stop.
52 %                  2 means failed: did not oscillate but also
53 %                  did not converge before hitting MAX_ITER.
54 %                  Caller can try with larger MAX_ITER
55 % pointsFound      n by 2 matrix, as in [x1 y1; x2 y2; .....]
56 %                  which contain coordinates of each point
57 %                  visited during steepestDescent the length is
58 %                  the same as number of points visited.
59 %                  This will be last point only if
60 %                  opt.accumlate=false
61 % levelSets        vector, contains the value of the objective
62 %                  function at each point. Last value of J(u)
63 %                  if opt.accumlate=false
64 % gradientNormTol  vector, contains the norm of gradient after
65 %                  each step. This will be last value only if
66 %                  opt.accumlate=false
67 % steps            vector. The optimal step used at each
68 %                  iteration, used golden section to find optimal
69 %                  step size.
70 %                  This will be last value only if
71 %                  opt.accumlate=false These are the alpha_k
72 %                  values.
73 % betaK            vector contains values of beta found at
74 %                  each step
75 %
76 % by Nasser M. Abbasi  ECE 719, UW Madison, HW 5
77
78 %pre-allocate data for use in the main loop below
79 N               = size(opt.u,1);
80 fLambda         = @(alpha,u,s) opt.objectiveFunc(u+alpha*s);
81
82 %collect data only if user asked for it.
83 if opt.accumulate
84     pointsFound     = zeros(opt.MAX_ITER,N);
85     levelSets       = zeros(opt.MAX_ITER,1);
86     gradientNormTol = zeros(opt.MAX_ITER,1);
87     steps           = zeros(opt.MAX_ITER,1);
88     betaK           = zeros(opt.MAX_ITER,1);
```

```
89  end
90
91  % initialize counters before main loop
92  k                    = 1;
93  currentPoint         = opt.u;
94  keep_running         = true;
95  status               = 0;
96  steps_in_oscillation = 0;
97  last_level           = 0;
98  current_grad         = opt.gradientFunc(currentPoint);
99  current_v            = -current_grad;
100
101 while keep_running
102
103     update_accumlate();
104
105     if k>1 && current_level>last_level% check for oscillation
106        if opt.stop_on_oscillation
107           steps_in_oscillation = steps_in_oscillation + 1;
108        end
109     end
110
111     check_convergence();
112
113     if keep_running
114
115        %A   = opt.hessian(currentPoint);
116        %lam = - dot(current_grad, current_v)/...
117        %                            (current_v.'*A*current_v);
118
119        %do not use norm on current_v here!
120        alpha = nma_golden_section(...
121         fLambda,currentPoint,current_v,0,1,sqrt(eps('double')));
122
123        % make step towards min
124        currentPoint = currentPoint + alpha* current_v;
125        last_grad    = current_grad;
126        current_grad = opt.gradientFunc(currentPoint);
127
128        %fletcher
129        %beta = norm(current_grad)^2/norm(last_grad)^2;
130
131        %polyak
132        beta = (current_grad.' * (current_grad-last_grad))/...
133                                                 norm(last_grad)^2;
134
135        current_v = -current_grad + beta * current_v;
```

```
136         if opt.accumulate
137            steps(k) = alpha;
138            betaK(k) = beta;
139         end
140
141         k = k + 1;
142      end
143  end
144  %done. Chop data to correct number of steps used before returning
145  if opt.accumulate
146      pointsFound     = pointsFound(1:k,:);
147      levelSets       = levelSets(1:k);
148      gradientNormTol = gradientNormTol(1:k);
149      steps           = steps(1:k);
150      betaK           = betaK(1:k);
151  else
152      pointsFound     = currentPoint ;
153      levelSets       = current_level;
154      gradientNormTol = current_grad_norm;
155      steps           = k;
156      betaK           = beta;
157  end
158
159      %-----------------------------
160      %internal function. Check if still need to keep iterating
161      function check_convergence()
162       % check if we converged or not
163       % Last check below can lead to termination too early for the
164       % banana function. Since at one point, J(u(k+1)) will get
165       % larger than J(u(k)) using bad step size. So it is
166       % commented out for now.
167          if k == opt.MAX_ITER || ...
168                    current_grad_norm <=opt.gradientNormTol || ...
169                  steps_in_oscillation>4
170              %let it run for 2 more steps to see the oscillation
171              %stop loop and set the status to correct reason
172              %why loop stopped.
173              keep_running = false;
174              if steps_in_oscillation>0
175                  status = 1;
176              else
177                  if k == opt.MAX_ITER
178                      status= 2;
179                  end
180              end
181          end
182      end
```

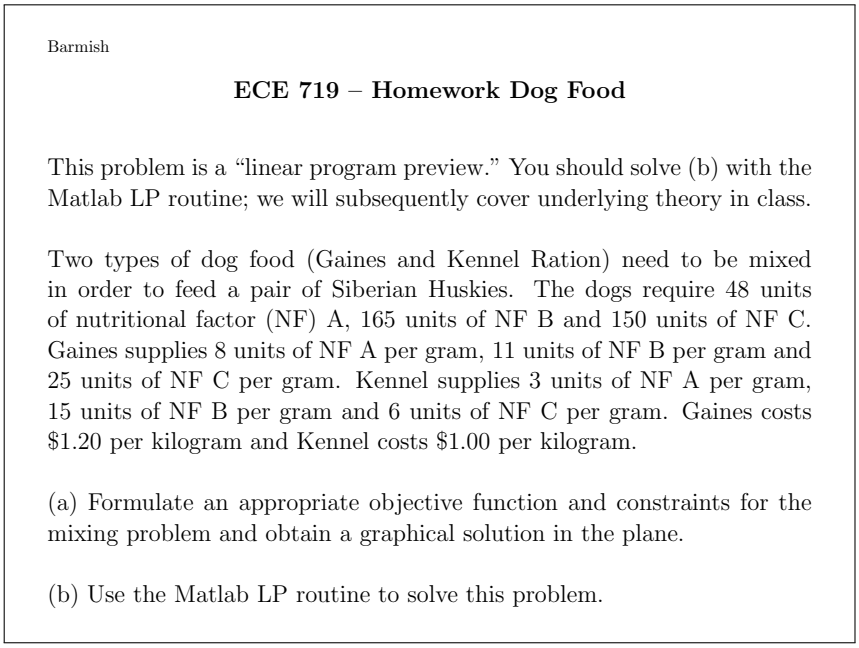
```
    %-----------------------------------------
    %internal
    function  update_accumlate()
        if k>1
            last_level = current_level;
        end

        current_level    = norm(opt.objectiveFunc(currentPoint));
        current_grad_norm = norm(current_grad);

        if opt.accumulate
            pointsFound(k,:)   = currentPoint;
            levelSets(k)       = current_level;
            gradientNormTol(k) = current_grad_norm;
        end
    end
end
```

```
%------------
%helper function to set plot attributes.
function nma_setMyLabels(varargin)

myXlabel = varargin{1};
myYlabel = varargin{2};
if nargin ==4
    myZlabel = varargin{3};
end
myTitle  = varargin{end};
h        = get(gca,'xlabel');
set(h,'string',myXlabel,'fontsize',10,'interpreter','Latex') ;

h = get(gca,'ylabel');
set(h,'string',myYlabel,'fontsize',10,'interpreter','Latex') ;

if nargin ==4
    h = get(gca,'zlabel');
    set(h,'string',myZlabel,'fontsize',10,'interpreter','Latex');
end

h = get(gca,'title');
set(h,'string',myTitle,'fontsize',10,'interpreter','Latex', ...
    'HorizontalAlignment','center') ;

set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
end
```

```
%====================================
%helper function to generate Contour data
```

```
function [X1,X2,Z] = nma_makeContourData(del,xlimits,ylimits)

x1      = xlimits(1):del:xlimits(2);
x2      = ylimits(1):del:ylimits(2);
[X1,X2] = meshgrid(x1,x2);
f1 = X2-X1.^3+5*X1.^2-2*X1-13;
f2 = X2+X1.^3+X1.^2-14*X1-29;
Z = f1.^2+f2.^2;
end
```

0.2 Problem 2



Barmish

ECE 719 – Homework Dog Food

This problem is a "linear program preview." You should solve (b) with the Matlab LP routine; we will subsequently cover underlying theory in class.

Two types of dog food (Gaines and Kennel Ration) need to be mixed in order to feed a pair of Siberian Huskies. The dogs require 48 units of nutritional factor (NF) A, 165 units of NF B and 150 units of NF C. Gaines supplies 8 units of NF A per gram, 11 units of NF B per gram and 25 units of NF C per gram. Kennel supplies 3 units of NF A per gram, 15 units of NF B per gram and 6 units of NF C per gram. Gaines costs $1.20 per kilogram and Kennel costs $1.00 per kilogram.

(a) Formulate an appropriate objective function and constraints for the mixing problem and obtain a graphical solution in the plane.

(b) Use the Matlab LP routine to solve this problem.

Figure 15: problem 2 description

0.2.1 part a

We need to minimize the cost of $48A + 165B + 150C$ by finding the optimal mix (quantities) of A, B, C obtained from Gaines and Kennel supply as shown in the following diagram

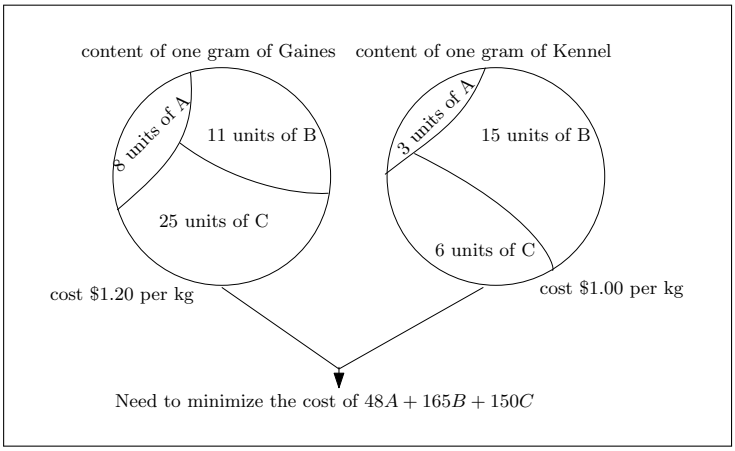


Figure 16: problem 2, part a

Let the amount (in grams) from Gaines be u_1 and let the amount of grams from Kennel be u_2. Therefore, we need to minimize

$$J(u) = (0.0012)\, u_1 + (0.001)\, u_2 \tag{1}$$

Since this is the cost. Since there are 8 units of A in each gram from Gaines, and there are 3 units of A in each gram from Kennel, then we have the first restriction which is

$$8u_1 + 3u_2 \geq 48$$

Similarly, we find for B and C the following

$$11u_1 + 15u_2 \geq 165$$
$$25u_1 + 6u_2 \geq 150$$

Convert to equality, and now use x instead of u since now we are converting to standard form

$$8x_1 + 3x_2 - x_3 = 48$$

Similarly, we find for B and C the following

$$11x_1 + 15x_2 - x_4 = 165$$
$$25x_1 + 6x_2 - x_5 = 150$$

Now we write the above in the standard form

$$\min c^T x$$
$$Ax = b$$

Or

$$\min \begin{bmatrix} 0.0012 & 0.001 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$

$$\begin{bmatrix} 8 & 3 & -1 & 0 & 0 \\ 11 & 15 & 0 & -1 & 0 \\ 25 & 6 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 48 \\ 165 \\ 150 \end{bmatrix}$$

A graphical solution was found by plotting the three constraints. Since the extreme point must be at a vertex of the feasible region, we see that it is at $u^* = (4, 8)$ which is confirmed using Matlab LP in the second part.

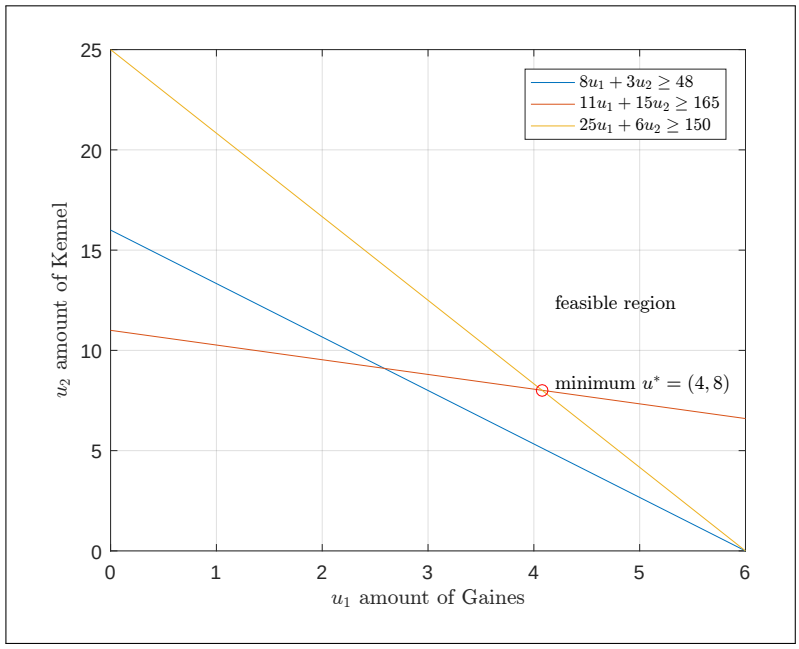


Figure 17: Graphical solution

Now contour lines are added to the above plot. Since the objective function is linear, the contour will be straight line, showing how $J(u)$ increases. The smallest value of $J(u)$ level set line which touches the first vertex of the feasible region will be the optimal point. Here is the result of the above plot, with contour lines added:

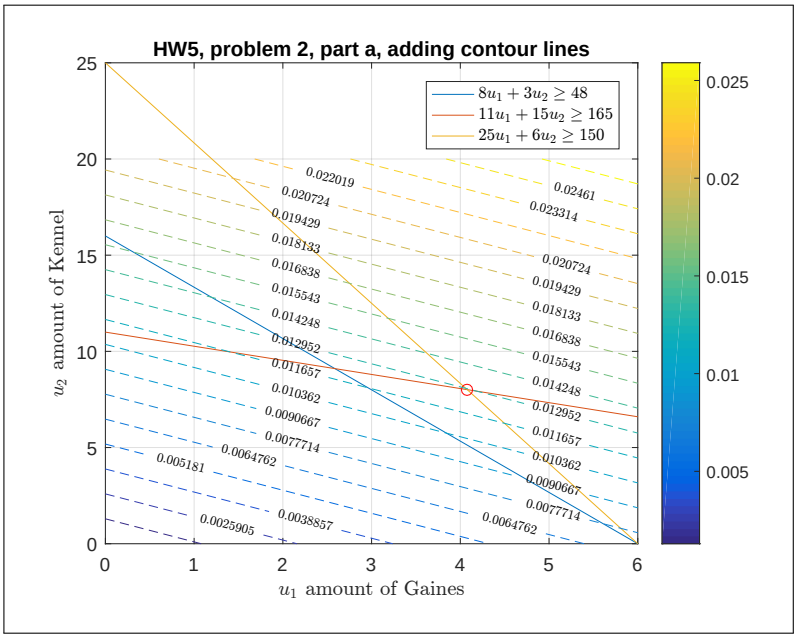


Figure 18: Graphical solution with contour lines added

```
clear; close;
x=0:6;
plot( x,(48-8*x)/3);
hold on;
plot( x,(165-11*x)/15);
hold on;
plot( x,(150-25*x)/6);
h=legend('$8u_1+3u_2\geq 48$','$11u_1+15u_2\geq 165$',...
         '$25u_1+6u_2\geq 150$');
set(h,'Interpreter','latex')
xlabel('$u_1$ amount of Gaines','Interpreter','latex');
ylabel('$u_2$ amount of Kennel','Interpreter','latex');
plot(4.077,8.0097,'ro');
text(4.2,8.3,'minimum $u^{\ast}=(4.077,8.0097)$',...
     'Interpreter','latex');
text(4.2,12.3,'feasible region','Interpreter','latex');
grid

%add contour lines
x1      = 0:0.1:6;
x2      = 0:0.1:20;
[X1,X2] = meshgrid(x1,x2);
Z = 0.0012*X1+0.001*X2;
[C,h]=contour(X1,X2,Z,20,'--');
clabel(C,h,'Fontsize',7,'interpreter','Latex');
title('HW5, problem 2, part a, adding contour lines')
colorbar
```

.

0.2.2 Part b

Matlab linprog was used to solve the above to find $x_1, x_2, x_3, x_4, x_5, x_6$. Here is the result for x^*

$$\begin{aligned} x_1 &= 4.0777 \\ x_2 &= 8.0097 \\ x_3 &= 8.6505 \\ x_4 &= 0 \\ x_5 &= 0 \end{aligned}$$

Mapping this back to u, we see that $u_1 = x_1$ and $u_2 = x_2$. Hence the minimum cost in dollars is from (1)

$$\begin{aligned} J(\boldsymbol{u}) &= (0.0012)\, u_1 + (0.001)\, u_2 \\ &= (0.0012)\, 4.0777 + (0.001)\, 8.0097 \\ &= 0.012903 \end{aligned}$$

The above is the cost of 4 grams from Gaines and 8 grams from Kennel. The above basically says to buy twice as much from Kennel as from Gaines.

0.2.3 Source code for problem 2

```
c1=0.0012;
c2=0.001;
f=[c1,c2,0,0,0];
A=[8,3,-1,0,0;
   11,15,0,-1,0;
   25,6,0,0,-1];
b=[48,165,150];
[X,FVAL,EXITFLAG,OUTPUT]=linprog(f,[],[],A,b,zeros(size(f)),[])
```

.

Result of above run

```
c1=0.0012;
c2=0.001;
f=[c1,c2,0,0,0];
A=[8,3,-1,0,0;
11,15,0,-1,0;
25,6,0,0,-1];
b=[48,165,150];
[X,FVAL,EXITFLAG,OUTPUT]=linprog(f,[],[],A,b,zeros(size(f)),[])
Optimization terminated.
X =
4.0777
8.0097
8.6505
0.0000
```

```
0.0000
FVAL =
0.0129
EXITFLAG =
1
OUTPUT =
iterations: 6
algorithm: 'interior-point-legacy'
cgiterations: 0
message: 'Optimization terminated.'
constrviolation: 8.5265e-14
firstorderopt: 4.0665e-10
```